A pattern-based Approach for an Early Detection of Popular Twitter Accounts

Jonathan Debure, Stephan Brunessaux, Camelia Constantin, Cédric Du Mouza

▶ **To cite this version:**

Jonathan Debure, Stephan Brunessaux, Camelia Constantin, Cédric Du Mouza. A pattern-based Approach for an Early Detection of Popular Twitter Accounts. International Database Engineering & Applications Symposium (IDEAS), Aug 2020, Séoul, France. 10.1145/3410566.3410600 . hal-02936179

HAL Id: hal-02936179

https://cnam.hal.science/hal-02936179v1

Submitted on 11 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Pattern-based Approach for an Early Detection of Popular Twitter Accounts

Jonathan Debure
AIRBUS & CNAM, Paris, France
jonathan.debure@airbus.com

Stephan Brunessaux
AIRBUS, Paris, France
stephan.brunessaux@airbus.com

Camelia Constantin
Sorbonne University, Paris, France
camelia.constantin@lip6.fr

Cédric du Mouza
CNAM, Paris, France
dumouza@cnam.fr

ABSTRACT

Social networks (SN) are omnipresent in our lives today. Not all users have the same behaviour on these networks. If some have a low activity, rarely posting messages and following few users, some others at the other extreme have a significant activity, with many followers and regularly posts. The important role of these popular SN users makes them the target of many applications for example for content monitoring or advertising. It is therefore relevant to be able to predict as soon as possible which SN users will become popular.

In this work, we propose a technique for early detection of such users based on the identification of characteristic patterns. We present an index, H^2M, which allows a scaling up of our approach to large social networks. We also describe our first experiments that confirm the validity of our approach.

CCS CONCEPTS

• **Information systems → Social networks**.

KEYWORDS

Twitter, popularity detection, pattern matching

ACM Reference Format:
Jonathan Debure, Stephan Brunessaux, Camelia Constantin, and Cédric du Mouza. 2020. A Pattern-based Approach for an Early Detection of Popular Twitter Accounts. In *24th International Database Engineering Applications Symposium (IDEAS 2020), August 12–14, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3410566.3410600

1 INTRODUCTION

Online social networks have become nowadays an essential means for communication, entertainment and marketing. Platforms like YouTube, Facebook, Twitter and Instagram gather hundreds of millions of users every day. While they have their own specifics and propose different content and interactions ways, these platforms share some common characteristics: first, their large number of users and the phenomenal amount of data (texts, pictures, videos, etc) produced daily; second, their network structure, with users connected to other users to share content; third, their high dynamicity with new users joining the platforms, others leaving, and connections between users which are continuously created or deleted.

These different characteristics make these platforms a tool particularly used to communicate information to a large number of people. In these networks, the most popular and influential users have quickly been the center of attention for many applications, since they will accelerate the spread of information to the greatest number of users [8]. For instance, for online advertising campaigns on social networks or on the Web, advertisers seek to place their advertisements among the users who have the most visibility in order to reach a maximum of people [2, 5, 9]. Likewise, for marketing purposes, highly followed users, called *influencers*, are paid to test and promote different products. In another area, popular users allow messages to be transmitted to a large audience for which social networks are the main media of information. These are the users who can quickly spread fake news or on the contrary bring a denial [7, 31]. Checking the content they publish is therefore particularly important. In the area of security, monitoring the content posted by some popular users who use social media for propaganda and / or indoctrination is also essential.

The various existing works offer techniques for detecting users who are already popular or influential in social networks. However, the various examples of applications presented above show that it is important to be able to identify the appearance of popular users on social networks as soon as possible. This article is, to our knowledge, the first to try to identify users who are on the way to more or less near future, to become popular. By detecting recurring patterns in the evolution of the popularity of accounts becoming popular, we manage with good precision to detect users several weeks before they become really popular. In addition, the index structure that we offer makes it possible to scale up to hundreds of millions of users and therefore allow our solution to be deployed for real social media platforms. Our experiences with real Twitter datasets validate our approach.

In summary, the contributions of our article are as follows:

(1) a characterization of the evolution of popularity for different classes of users (popular, non-popular, becoming popular);
(2) a pattern-based approach for early detection of popular users;
(3) an indexing structure for an efficient pattern-matching which scales to hundreds of millions of users to an early detection of future popular users;
(4) a validation on a large real Twitter dataset.

IDEAS 2020, August 12–14, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *24th International Database Engineering Applications Symposium (IDEAS 2020), August 12–14, 2020, Seoul, Republic of Korea*, https://doi.org/10.1145/3410566.3410600.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 describes the data model for popularity evolution. An analysis of a real Twitter dataset and the patterns we extracted for different classes of users is presented in Section 4. We introduce our pattern-based approach along with its indexing structure for an early detection of users becoming popular in Section 5. Section 6 gathers some of the experiments we perform to validate our approach. We conclude the paper and introduce some future work in Section 7.

2 RELATED WORK

Most of the recent work trying to estimate the popularity of accounts in social networks have focused on influence spreading ([18], [21], [20], [19]) as a measure for popularity. Basically these works observe how many users along with their distance (number of intermediate users between the user who produces the news and the one who receives them) will receive an item produced by a user. They rely consequently on the topology and on the eventuality (based on his homophily, his topics of interests, his activity, etc) for a user to propagate this item. Some works try not only to study the propagation but also to capture and to estimate the ability of some users to trigger actions or change an opinion in the network, generally their direct neighborhood. Different influence studies propose to score users ([1],[27],[28]) to rank them and to highlight principal node of a social network.

Users popularity analysis in social media, especially in social networks, has taken importance in the last decade because they appear to be a rather precise way to estimate the opinion of a pool of users. For instance, during American elections, online social media have been used by politician to spread their program [17]. Different analyses of the social networks try to determine the most popular candidate and to predict the elections results [10], [15]. Similarly, social media analyses try to predict future popular content [26], like hot news [6] and try to understand hidden mechanisms. Multiple studies try to find how information are spread on Twitter, with probabilistic model [30] and others study the retweet action from Twitter [14] as an indicator of users popularity. In [24], the authors explain that retweet is time-sensitive. The main difficulty for these approaches is that they assume we have a knowledge of the content published or liked by any user, and also enough information to estimate their opinion or interests which is most of the time either very costly or impossible to get in real social network systems, except for the platform owners.

A solution for popular user detection and prediction, that we adopt in this work, is to identify features which characterize popular users, based on a large sample of users, and to perform machine learning techniques to identify other popular users on the global dataset and/or to perform predictions to early detect their apparition. We propose in this paper to extract patterns which characterize a class of popularity for users. Patterns mining is a popular method to find frequent occurring patterns. Patterns mining is divided in two majors models: item-set mining and string-mining. Item-set mining focuses on detecting frequent item-set and is mostly used in database mining. The two most popular algorithms for item-set mining are APRIORI [25] and FPGrowth [13]. APRIORI extracts all item-sets of a specific length respecting a minimum support, and then scans transactions at each iteration that makes this algorithm rather slow. FPGrowth is an improved APRIORI algorithm which relies on a Tree (FP Tree) to mine frequent item-set and which only needs to scan database twice. Item-set mining is also used for association rules mining [4], *i.e.* the extraction of correlation between items. String-mining is a distinct pattern mining technique, which consists in detecting frequents patterns in alphanumerical sequence generally very long. This technique is mainly use in bioinformatics[3] to analyze DNA/RNA and to detect or to extract proteins from nucleotide sequences. String mining is mostly represented by three algorithms: SPADE [29] which is a vertical sequential patterns mining, FreeSpan [11] which partitions the search space and projects sequences and PrefixSpan [12] that is based on FreeSpan but avoids to check every possible combination.

3 THE DATA MODEL

We introduce in this section our notations and our data model. We consider the Twitter platform and its underlying directed graph $(\mathcal{U}, \mathcal{F})$ where $\mathcal{U}$ denotes the set of nodes, *i.e.* users, and $\mathcal{F} \subseteq \mathcal{U}\times\mathcal{U}$ is the set of edge, such as $(u_1, u_2) \in \mathcal{F}$ means user u_2 follows user u_1.

3.1 Account popularity

Remind that our objective is to perform early detection of (future) popular users. Assuming the existence of the function $Follow : \mathcal{U} \times [0, T] \rightarrow \mathbb{N}$ which returns the number of followers for an account $u \in \mathcal{U}$ at the instant $t \in [0, T]$, we adopt in this paper the following definition of popularity:

Definition 1. *[Popularity] The popularity of an account corresponds to its visibility, that means how many persons can read, comment, propagate a message that this account produces. It is here simply estimated by the number of followers $Follow(u, t)$ an account u possesses at the instant t.*

With respect to this definition we propose the following account classification:

- Non-popular accounts: this class regroups users that are never popular all along the observation period $[0, T]$. So, assuming an unpopularity threshold φ, these accounts verify that: $u \in \mathcal{U}, \forall t \in [0, T] : Follow(u, t) \leq \varphi$
- Popular accounts: this class corresponds to users that are already popular on our study period $[0, T]$. So assuming a popularity threshold ε, this implies that at $t = 0$ we have $Follow(u, 0) \geq \varepsilon$.
- Becoming popular accounts: this class regroups users that are not popular at the beginning of our period $[0, T]$ but are popular at the end. So, based on our two thresholds ε and φ, it corresponds to users $u \in \mathcal{U}$ such as $Follow(u, 0) \leq \varphi$ and $Follow(u, T) \geq \varepsilon$.

3.2 Popularity evolution

We assume that the platform periodically updates its statistics for each user. The period between two updates is constant and is considered in the following as our indivisible time unit. So at each

time instant $t \in [0, T]$, we report for each account u the number of followers. To estimate the popularity evolution, we compute the gain in number of followers between two time instants. To reduce the impact of the size of the accounts, we propose to use the log function on the gain since a gain of 10k followers should be considered as a gain of the same order as a gain of 20k followers when the user has a popular account. Due to the domain definition of the log function we use the following definition for the *gain* function.

Definition 2. *[Popularity gain] Consider an instant $t \in [0, T-1]$, the popularity gain for a user $u \in \mathcal{U}$ is:*

$$gain(u,t) = \begin{cases} \log(Follow(u,t) - Follow(u,t-1)) \\ \qquad if Follow(u,t) > Follow(u,t-1) + 1 \\ 0 \quad if |Follow(u,t) - Follow(u,t-1)| \leq 1 \\ -\log(Follow(u,t-1) - Follow(u,t)) \\ \qquad if Follow(u,t-1) > Follow(u,t) + 1 \end{cases}$$

Consequently, the popularity evolution for an account over a given period $[0, T]$ corresponds to a time series made up of the popularity gain for each time unit. We formally adapt the following definition for the popularity evolution.

Definition 3. *[Popularity evolution] The popularity evolution of user $u \in \mathcal{U}$ on the time period $[0, T]$ is represented by the time series*

$$\pi(u) =< gain(u,1), gain(u,2), \ldots, gain(u,T) >$$

.

We denote with Π the set of the popularity evolution for all users from $\mathcal{U}$.

This raw time series is interesting to extract some statistics about the given dataset but is not suitable to compare users in order to cluster them and to detect classes of users with some specific behaviors, or oppositely to identify users who have a divergent behavior. In order to achieve these goals, a traditional approach (see for instance the SAX approach [16, 22]) consists in encoding the time series using a limited set of symbols.

So we assume the existence of an alphabet of symbols Ω for the encoding and of a mapping function $mapping : \mathbb{R} \rightarrow \Omega$. Defining the size of the alphabet and the mapping function is a difficult task which must highly rely on the properties of the studied dataset. The size of the alphabet used for the encoding sets a level of refinement for the encoded sequence. Indeed, a large alphabet will provide more precision about the popularity evolution, but it reduces the number of similarities between sequences detected. Oppositely, a small alphabet will lead to the extraction of numerous similar subsequences between the different sequences, while they correspond in reality to an evolution relatively different. Similarly the mapping function will highly impact the similarity detection between sequences. When too many different gain values are mapped to the same symbols, while other symbols correspond to very few gains, it results in an issue similar to the use of a small alphabet: similar subsequences that are detected may correspond to very different behaviors. So the choice of the alphabet and the mapping function has an important impact on the results. We do not investigate further this problem, but we take it into consideration when proposing our alphabet and mapping function in our analysis and experimental sections.

Based on this representation, we want to study whether some subsequences, we will call *patterns* in the following, are characteristic of a popularity class. If such subsequences exist, we expect them to allow us to perform early detection of emerging popular accounts.

Assume a *contains* function, $contains : 2^{\Omega} \times \mathcal{S} \rightarrow \mathbb{N}$, with 2^{Ω} denoting the powerset of Ω, where $contains(x, y)$ returns 1 the sequence y contains the word x, and 0 otherwise. Then we adopt the following definition for a popularity pattern.

Definition 4. *[Popularity pattern] Consider a given size value σ and assume the existence of a relevance threshold Γ, a popularity pattern of size σ is a word $p \in \Omega^{\sigma}$ such that*

$$\frac{\sum_{s \in \mathcal{S}} contains(p, s)}{|\mathcal{S}|} \geq \Gamma$$

The relevance threshold Γ allows to set a minimal support for a pattern. In other words, we only keep patterns which are significant because they are enough present in several sequences.

We denote $\mathcal{S}_{\Pi}$ the set of all the popularity patterns for a popularity evolution set Π. The sets of *exclusive* popularity patterns for the restrictions to the popular, unpopular and becoming popular evolution sets are denoted $\mathcal{S}_{\Pi^+}$, $\mathcal{S}_{\Pi^-}$ and $\mathcal{S}_{\Pi^*}$ respectively. By *exclusive* we mean the popularity patterns which are present only in the considered class of users.

4 POPULARITY ANALYSIS

This section aimed at analyzing a real Twitter dataset to check the existence of such characteristic evolution patterns in our three classes of users. We first introduce our dataset along with its main features, then we extract the popularity patterns for each class and we analyze these different sets of patterns.

4.1 Presentation of our dataset

To build our dataset we use the Twitter API Stream that allows us to collect 1% of all tweets published on the platform. For our dataset, we only collect users metadata and we select users that had a sufficient activity, *i.e.* at least 3 tweets, during our observation period of 36 weeks. We obtain a dataset consisting of around 32.9M users along with 150M tweets.

Then we set the two thresholds φ and ε necessary to determine our three groups of users according to their popularity. Based on the follower distribution of our dataset depicted in Figure 1, we decide arbitrarily that an account with less than $\varphi = 400$ followers is considered as unpopular, while an account with more than $\varepsilon = 2,000$ is identified as a popular account. Thus, the class of *becoming popular* accounts corresponds to the accounts which have less than 400 followers at the beginning of our observation period and more than $2,000$ at the end. We report in Table 1 the size of each class of accounts in our dataset that we will investigate in the following. As expected, most accounts belong to the non-popular class and present a low activity with 3-4 tweets generally on the period of observation. The 2.1 million popular accounts have a more important activity, but there is an important discrepency between accounts as it is enlightened by a high standard deviation. In fact, this class is quite heterogeneous with for instance news agency

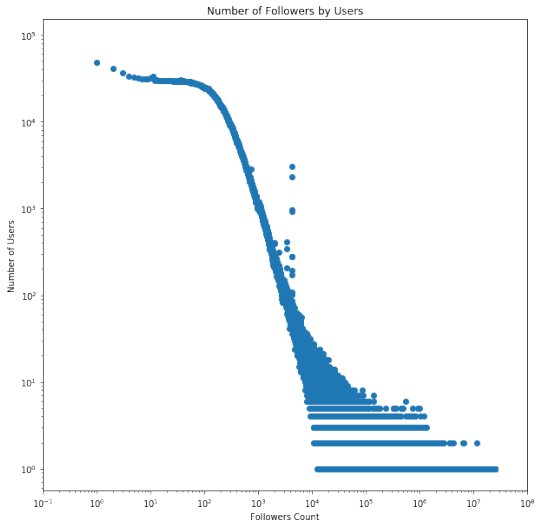

Figure 1: Follower distribution

Table 1: Sub-datasets for each class of users

	global	non-popular	becoming pop.	popular
# accounts	3.2289e+07	3.0106e+07	7.7364e+04	2.1056e+06
# tweets	1.4984e+08	9.2258e+07	3.6330e+06	5.3951e+07
tweets: mean	4.64	3.06	46.96	25.62

or brand accounts with an important activity, and popular personalities (actors, singers, etc) with many followers but few tweets. Accounts that are not popular at the beginning of the observation period but then become popular are not as numerous, but they have a higher activity. This can somehow explain that they become popular on a specific topic because they publish more on this topic what attracts followers.

Table 2 shows the number of followers for each class of users. In our dataset the average number of followers is 1,428, which is more than the last published estimation [1] due to our 3-week threshold for the user activity which discards accounts with few followers and extremely low activity (especially remember that we use the Twitter API with only 1% of the messages). We observe that non-popular accounts have a quite stable number of followers. Oppositely, we see that while becoming popular accounts and popular have generally between 2.10^3 and 3.10^4 followers, the last decile of users reaches respectively millions and dozens of millions of followers, with a more important discrepancy for popular accounts.

4.2 Pattern extraction

In order to perform our pattern extraction, we have first to set the size of the alphabet Ω and to determine the *mapping* function used for the popularity evolution encoding. As explained above, the *mapping* function should, as much as possible, uniformly distribute the gain values on the different symbols of Ω.

We compare different sizes of alphabet for Ω and we finally choose $|\Omega| = 8$. Remember that a small alphabet do not allow

[1] https://www.brandwatch.com/blog/twitter-stats-and-statistics/

Table 2: Followers for each class of users

	global	non-popular	becoming pop. at t=0	becoming pop. at t=T	popular
mean	1.4280e+03	1.2737e+02	1.4059e+02	8.2574e+03	2.2431e+04
std	5.2869e+04	1.1283e+02	1.2261e+02	4.9651e+04	2.3761e+05
min	2.4000e+01	0.0000e+00	0.0000e+00	2.0010e+03	2.0010e+03
25%	5.2000e+01	2.8000e+01	2.7000e+01	2.4420e+03	2.9330e+03
50%	1.8800e+02	9.6000e+01	1.1050e+02	3.3160e+03	4.7430e+03
75%	4.8800e+02	2.0800e+02	2.4000e+02	5.9640e+03	1.0699e+04
85%	8.1900e+02	2.7000e+02	3.0100e+02	9.4710e+03	1.8553e+04
90%	1.2040e+03	3.0700e+02	3.3300e+02	1.2800e+04	2.8601e+04
max	7.7129e+07	3.9900e+02	3.9900e+02	6.7735e+06	7.7129e+07

symbol	A	B	C	D	E	F	G	H
range	$]-\infty,-2]$	$[-2,-0.7]$	$[-0.7,0.7]$	$[0.7,1.6]$	$[1.6,2]$	$[2,2.7]$	$[2.7,3]$	$[3,\infty[$

Table 3: Popularity evolution encoding

Table 4: Symbols distribution

symbol	Becoming Pop.	Popular	Non-popular	Global
A	3.94%	3.39%	0.0%	1.39%
B	7.20%	16.50%	0.60%	4.94%
C	6.74%	20.27%	86.7%	59.14%
D	14.72%	29.88%	12.30%	16.05%
E	18.36%	12.02%	0.33%	5.91%
F	36.55%	12.08%	0.05%	9.15%
G	6.64%	2.58%	0.0%	1.72%
H	5.82%	3.24%	0.0%	1.69%

to capture significant patterns since they will cover very distinct behaviors, while a large alphabet provides very precise patterns but these patterns only correspond to a very small number of sequences.

We report in Table 3 our implementation of the *mapping* function for the computed gain values according to Definition 2.

Then we compute the gain of followers for each week of the 36-week observation period and we apply the *mapping* function for the encoding. Due to the low capture rate of the Twitter API (only 1% of the tweets) and the non-uniform publishing behavior of the users, several values are missing in many popularity evolution sequences. Consequently, we decide to apply linear interpolation to fill small gap of maximum two missing values. Users with sequences presenting gaps of more than 2 symbols are discarded from our dataset. Table 4 presents the symbol distributions for the different datasets. We observe that non-popular accounts have as expected a zero or very moderate growth with 86.7% of C-symbol, *i.e.*, a gain or loss of maximum 5 followers. We note that they also have very few follower losses (0.6% of B-symbol which represents a loss of 5 to 100 followers), and are therefore very stable. For the popular accounts, the evolution is more varied: they can show significant growth, stagnate or have a significant decrease. Significant decreases can have several reasons: a sudden unpopularity due, for example, to a questionable decision-making, or else identification as a false account or an account having bought followers. Finally, accounts that become popular generally have rather long periods of significant growth, which explains our observation of around 49% of F, G and H symbols, so a gain of more than 100 followers.

Table 5: Number of patterns found with a min. support of 1%

pattern length	length 3	length 4	length 5
non-popular	11	18	27
popular	123	144	154
becoming popular	182	194	165
becoming popular - (non-popular + popular)	70	75	69

Traditional pattern extraction algorithms return item-set of symbols that are not necessarily closed. For our matching model, we need to extract only patterns that contains symbols which follow each others without gap. To execute a pattern extraction algorithm which take only frequents item-sets of consecutive symbols, we perform a pattern extraction on our 3 datasets using a sliding windows approach and reporting all sequences of k-symbols encountered. We decide to only keep patterns with a support greater than 1%. We report in Table 5 the number of patterns we found. We observe that the non-popular class is characterized by a low number of patterns. There are two reasons for this result: first, non-popular users exhibit a number of followers which does not vary in an important way, so most of the symbols for their popularity evolution are C or D symbols. Moreover, the class of non-popular users is very large, so a minimum support of 1% implies that this pattern is present in a very large number of popularity evolution sequences. With a support of 0.5%, the number of patterns is 286, 1051 and 2782 for respectively length 3, 4 and 5. The classes of popular and becoming popular users have approximately the same number of patterns, between 100 and 200. Observe that the size of the extracted patterns have a double impact on the number of patterns. Indeed, a pattern of k symbols with a support greater than 1% can provide potentially 2 or more patterns with $k+1$ symbols with a support greater than 1%. But oppositely, it can provide patterns with a support lower than 1%. This explains for instance why the number of patterns increases from 182 for size 3 to 194 for size 4 for the becoming popular dataset, and then decreases to 165 for size 5. When comparing the patterns found in the becoming-popular class to patterns from other classes, we see that several patterns are present in several classes. Nonetheless, we exhibit around 70 exclusive patterns that we will use for our detection.

We report in Figure 2 the support for the top-100 most frequent patterns that we extracted for each dataset. We first observe that regardless of the data set or the size of the extracted patterns, the frequency distribution of the patterns follows a power law. Patterns for the becoming-popular class have a more important support that those of other classes. For a size 3, the most frequent pattern is present among 55% of the sequences, the tenth most frequent among 15% and the fiftieth is still present in around 5% of the sequences. It results that the popularity evolution of becoming popular users is characterized by several dozens of patterns which can be used consequently to identify users from this class. The support for popular patterns is less important but remains significant: the most frequent pattern is present among 30% of the sequences, the tenth most frequent among 8% and the fiftieth is still present in around 3% of the sequences. For this class, we consequently observe the existence of a large number of characteristic patterns. However, as noticed in Table 4, the distribution of the symbols is less biased than the one of becoming popular users, what leads to more patterns but

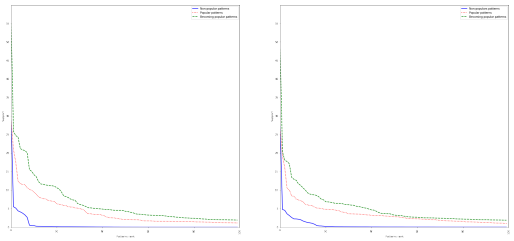

Figure 2: Support for 3 (left) and 4 (right) -symbol patterns

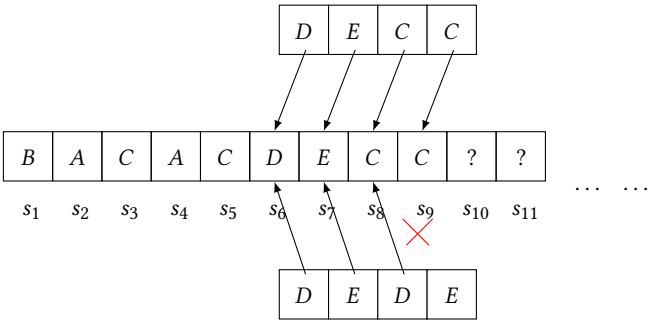

Figure 3: Example of matching attempt for patterns DECC and DEDE on a popularity evolution sequence

with less support. Finally the non-popular class is characterized by a small number of patterns with an important support: the most frequent pattern is present among 30% of the sequence, the tenth most frequent among 1%. Finally, we observe a long tail of patterns with a support lower than 1%. Two parameters explain this observation: first, the sequences of non-popular users are rather short since they have a low activity and we stop the evolution sequences if two or more symbols are missing, which is quite uncommon with popular or becoming popular users. Second, their number of followers is quite stable, as we can see in Table 4 with 86.7% of C-symbol. The results for a size 4 of pattern are similar, except we observe the power-law curves are a bit smooth compare to size 3. In fact, as explained above for Table 5, we have for the different classes fewer patterns with an important support but more with a medium support.

5 USING PATTERNS FOR AN EARLY DETECTION OF POPULAR USERS

Once we have identified the different sets of *exclusive* popularity patterns for the restrictions to the popular, non-popular and becoming popular evolution sets are denoted $\mathcal{S}_{\Pi^+}$, $\mathcal{S}_{\Pi^-}$ and $\mathcal{S}_{\Pi^*}$ respectively, we intend to use them to identify users becoming popular before they reach the popularity threshold ε.

5.1 Popularity pattern matching

Our objective is to test the matching of any pattern $p \in \mathcal{S}_{\Pi^*}$ with any popularity evolution sequence $s \in \mathcal{S}$ whenever it increases with a new symbol.

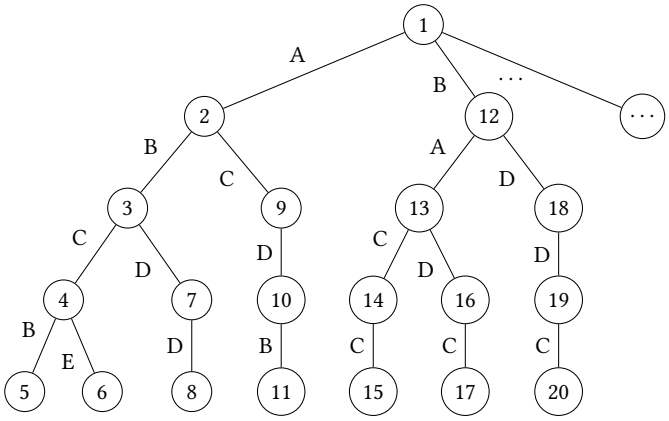

Figure 4: Tree structure for D_{pop} = {*ABCB*, *ABCE*, *ABDD*, *ACDB*, *BACC*, *BADC*, *BDDC*}

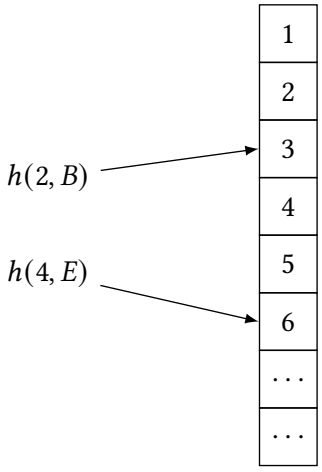

Figure 5: Our hash-based implementation

Example 1. *Figure 3 depicts an example of matching attempt. For our user, we report a new symbol C at week* 9 *corresponding to his popularity gain between weeks* 8 *and* 9. *We must then try to match each pattern of our whole set of* becoming popular *patterns with the suffice of the popularity evolution sequence. The pattern, DEDE does not match the suffice, oppositely to the pattern DECC. So we report a match and we consider the user corresponding to this popularity evolution sequence as a possible future popular user.*

So basically, our problem is a multi-stream multi-pattern matching issue. The difference with traditional pattern matching solutions (see Section 2) is that we have to deal with hundreds of millions of users and hundreds of patterns. This raises an important scalability issue. Our objective in the following is to propose a structure for a fast multi-pattern matching on a very large set of streams.

5.2 The H^2M index

To perform the matching of a set $\mathcal{S}_\Pi$ of patterns over a sequence π of symbols, several approaches are proposed in literature. Most efficient ones rely on finite state automaton (FSA). However, a traditional FSA presents some loops which results in testing for each state reached whether it corresponds to the final state of a given pattern or not. To limit this number of tests, we propose to rely on a trie representation, in other words a tree-shaped deterministic finite automaton. Since we make the assumption that our popularity patterns have a fixed size σ, it means that all (1) the paths from the root to a leaf have a length of σ and (2) only leaves correspond to the the final symbol of a popularity pattern.

Example 2. *Figure 4 is an example of the tree structure for the set of patterns { ABCB, ABCE, ABDD, ACDB, BACC, BADC, BDDC}. Here* $\sigma = 4$ *so we can check that all paths to a leaf have a length of* 4 *and each leaf corresponds to one pattern.*

This fixed-size of patterns allows us to consider a sliding window on the different sequences with only the σ last symbols which are used for the matching attempts.

Since we consider applications with hundreds of millions of users, we need to evaluate the transitions in a very efficient way. Thus we propose to choose the hash-based implementation for our pattern tree structure. So formally our pattern tree is defined thanks to our Pattern-Tree Hash index (PTH-index) as:

Definition 5. *[Pattern-Tree Hash index] A pattern-tree hash index PTH is defined on a pattern set* $\mathcal{S}_\Pi$ *as a couple* (V, h_{trie}) *where* V *is a set of nodes* $v = (id, isLeaf) \in V$ *with* id *a node id and* $isLeaf$ *a boolean set to* $true$ *when this is a leaf node, and* $h_{trie} : V \times \Omega \rightarrow V$ *is a hash function which represents the edges with the following properties:*

i) h_{trie} *is injective, so each node could only have one parent (except the root node),*
ii) *if* $\forall v \in V, v.isLeaf = true \Rightarrow \exists (x_1, x_2, \ldots, x_\sigma) \in \Omega^\sigma$, $h_{trie}(h_{trie}(\ldots(h_{trie}(root, x_1), x_2), \ldots, x_\sigma) = v \wedge x_1.x_2.\ldots.x_\sigma \in \mathcal{S}_\Pi$

Example 3. *Figure 5 represents the hash-based structure corresponding to the tree structure of the Figure 4. For instance with the symbol E, the node whose id is* 4 *leads to the node whose id is* 6.

For any incoming symbol for a given sequence π, we use this trie and we determine the new positions reached in this one. The following proposition determines the number of position in the trie we have to store for any user.

Proposition 1. *[Number of stored positions] Since the depth of the trie corresponds to the length of the patterns,* σ, *the different suffices with length in* $[1, \sigma]$ *must be considered when a new symbol is added and each suffice could reach a position in the trie. So at any time we have :*

- *to record for next matching attempt the different positions reached with the suffices with length in* $[1, \sigma - 1]$,
- *to report potentially a match when reaching a leaf Consequently the space requirement for storing user information is* $O(\sigma)$.

To efficiently retrieve the different positions in the trie for a given user, we propose to rely on a second hash structure. So consider the set $\mathcal{I}$ of the user identifiers. We then define our *Automaton Positioning Index* ($AP - index$) which consists of entries.

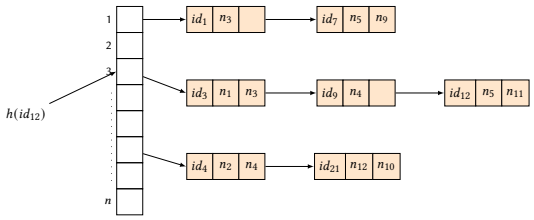

Figure 6: Our hash-based implementation

Definition 6. *Let $id \in \mathcal{I}$ a user identifier and V denotes the set of nodes ids from the trie. The entry indexing P, denoted $P(id)$, is a tuple (id, pos) with $pos \in 2^{V^\sigma}$ (the powerset of V^σ) a set of positions in the automaton.*

The indexing is "dense", *i.e.*, every popularity evolution sequence in the database is indexed, as soon as a user appears in the system, and by a different entry. $AP-Index$ is a hash file, denoted $AP[0..L-1]$, with directory length L (Fig. 6). Building a hash file with a hash entry for each user will lead to an extremely large index, since it requires $|\mathcal{S}|$ memory blocks and thus the index structure could not fit in memory. Consequently, the elements of AP refer to buckets or lines, each containing a list of entries. Each $AP[i]$ contains the address of the i-th bucket. Since entries have a similar size, *i.e.*, a user id and a set of positions with a number of elements between 1 and $\sigma-1$, we can store in a block with size B between $B/(|id|+(\sigma-1).|p|)$ and $B/(|id|+|p|)$ entries ($|id|$ and $|p|$ denote respectively the size of a user id and an automaton node id). So we set the size of the list accordingly. We consider in the following the pessimistic approach where all entries inside a bucket have $\sigma-1$ positions stored. We also assume we have $|\mathcal{S}|$ users to index. We explain below how to manage the dynamicity of the system, with users who join or leave frequently. With these settings, we can set the value of L:

$$L = |\mathcal{S}| \times \frac{|id| + (\sigma-1).|p|}{B}$$

Statistics over the entry size (so basically, what is the average number of automaton positions stored for a user) could permit to propose a L value with a higher space gain.

All together, the $AP-Index$ line structure is similar to a posting list in an inverted file. Since the key used for hashing are user ids, it is easy to design a hash function that evenly distributes the entries in the buckets, at least when the index is built. To manage the dynamicity of the users set, we could have several strategies which could moreover be combined. First, we could adopt the strategy used in database systems for storing the data, *i.e.* not to fill the data block (the *PCT-free* parameter). Thus, we could for instance choose a higher L value when creating the index and to have consequently some free space in each block to add new entries. In addition, since we use a hash file, lines should have a collision resolution method such as classical separate chaining that uses pointers to an overflow space. Such a technique accommodates moderate growth, but if we need to accommodate large growth, then we need a dynamic hashing method such as linear hashing [23].

The combination of our two hash-based structures for an efficient matching, $PTR-index$ and and $AP-index$, composes our proposal we name H^2M -index.

Table 6: Quality of the detection

	precision	recall	F1
Global dataset	0.7260	0.7604	0.7428
Removing popular users	0.9972	0.7604	0.8629

6 EXPERIMENTS

All experiments have been realised on a dedicated machine with 8x Intel® Xeon® Processor E7-4830 v2 (80 cores) and 512 Gb RAM. We have chosen Python for our development. To validate our approach, we first evaluate the quality of our detection, then we compare the performances of our matching structured with other implementations.

6.1 Quality of the detection approach

To estimate the quality of our approach, we split our global dataset of $32M$ users in a training set with 80% of the users, and a test set with the remaining 20%. We divide our training test into 3 groups of users according to their popularity evolution as explained in Section 4 and we perform our pattern extraction process on each of these user datasets. Tables 1, 2 and 5 describe the characteristics of these datasets along with their number of patterns.

Then, we try to match the patterns of the different classes on our test dataset to detect respectively non-popular, popular and becoming-popular users. Due to space limitation, we only present here the matching of becoming-popular patterns. We report results in Table 6. We observe that we have an overall precision of 0.7260. However, this precision can be largely improved by a fast pre or post processing by discarding popular users. In fact, we can at any time stop following a user that is identified as popular, *i.e.* those having at any time more than 4,000 followers. When discarding these users, we reach a precision of 0.9972. The recall reaches 0.7604, so a good $F1$ value of 0.8629. We achieve a higher recall when reducing the minimal support during the pattern selection. If we keep patterns with a lower support, we achieve a 0.85 recall, but the precision drops to 0.75 because these patterns are less characteristic of the associate class and consequently some sequences of other classes may also match.

Since our objective is to detect future popular users before they become popular, everytime a matching occurs, we measure the number of weeks between the week of matching and the week during which they actually become popular. We report the ratio of becoming popular users detected with respect to the number of weeks in advance that this user is detected actually popular in Figure 7. We observe that in 80% of cases, our approach allows us to detect a popular user at least 1 month before they actually become popular, and in 60% of cases, at least 2 months before. This rate is still around 40% for detection at least 3 months in advance. This confirms the interest of our approach and its effectiveness.

6.2 Scalabity

Finally, we perform experiments to study the scalability of our index and matching compared to existing matching structures. As competitors we implement:

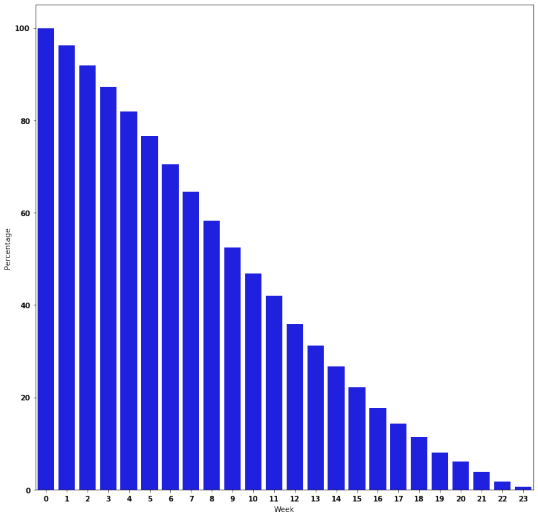

Figure 7: Number of weeks in advance a popular user is detected

- **SimpleTree**: implementation of a tree in standard python library with the possibility to search if a sub-sequence is in the tree. For each user we store in a hashmap the $\sigma - 1$ last symbol read (σ is the pattern size).
- **FSA**: we have use Automata-lib [2] which implements the structures and algorithms for finite automata.
- H^2M: our solution introduced above.

We generate according to the symbol distribution observed in our real dataset (see Table 4) datasets of respectively 1M, 10M and 50M popularity evolution sequences of 52 symbols (to simulate 1 year). We measure time to evaluate an incoming symbol for a user with the different structures along with the memory requirement to store information for the respectively 1M, 10M and 50M users. We report in Table 7 our results. We first observe that H^2M index allows to get the best performances for the matching since a new incoming symbol can be processed in $0.2ms$ when managing 1M users, so a gain of 90% and 250% with respectively *SimpleTree* and *FSA*. This matching time increases with the pattern size. We notice a 40% increase of the matching time for all structures. The rationale is that we have a 4-symbol subsequence in *SimpleTree* or *FSA*, or 3 positions in our structure, to consider instead of 3. Regarding the number of users to manage, we observe a constant matching time as expected, since we adopt a dynamic extension of the hashmap for the different structures to keep the current states for each users to avoid collisions. Consequently, the treatment of an incoming symbol is constant. We can also observe that our implementation requires a similar memory space compare to FSA while it saves 24% space compare to SimpleTree. The rationale is that we have to keep the $\sigma - 1$ last symbols for any user for the SimpleTree while we keep only 1 to $\sigma - 1$ reachable states for a user with H^2M. The memory gain increases with the pattern size, and we reach 44% gain for patterns with size 4.

[2] Automata-lib 3.1.0 : https://pypi.org/project/automata-lib/

Table 7: Performances of the matching

	Patterns size :	Size 3		Size 4	
Type	Nb of users	Time (ms)	RAM (MB)	Time (ms)	RAM (MB)
SimpleTree	1M	0.57	275.05	0.82	319.54
	10M	0.58	2526,00	0.81	3238.62
	50M	0.56	15067.56	0.80	16161.63
FSA	1M	1.05	205.86	1.53	227.55
	10M	1.05	2152.37	1.50	2199.18
	50M	1.03	11570.12	1.48	11962.11
H^2M	1M	0.30	214.97	0.41	222.32
	10M	0.30	2120.89	0.42	2197.58
	50M	0.31	11568.76	0.42	11954.67

7 CONCLUSION

This paper proposes a solution to tackle the early detection of future popular users. It is based on a characterization of users who become popular using popularity evolution patterns. Our analyzes show the existence of such patterns and our experiments confirm that they make it possible to detect future popular users several weeks before they are actually popular. We also offer a structure to allow scaling up of matching to millions of users.

We have several perspectives to complete this work. First of all we wish to make a more detailed analysis of the popularity evolution in order to determine the alphabet which will then allow a better detection. In addition, we want to look at other criteria (replies, quotes, likes, message propagation, etc.) and not only at simply the number of followers in order to identify influencers rather than popular users.

REFERENCES

[1] Klout score: Measuring influence across multiple social networks. In *IEEE Intl. Conf. on Big Data (Big Data)*, pages 2282–2289, 2015.
[2] Zeinab Abbassi, Aditya Bhaskara, and Vishal Misra. Optimizing Display Advertising in Online Social Networks. In Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi, editors, *Proc. Intl. Conf. on World Wide Web, (WWW) 2015*, pages 1–11. ACM, 2015.
[3] Mohamed Abouelhoda and Moustafa Ghanem. String mining in bioinformatics. In *Scientific Data Mining and Knowledge Discovery*, pages 207–247. Springer, 2009.
[4] Rakesh Agrawal, Tomasz Imieliundefinedski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.
[5] Çigdem Aslay, Wei Lu, Francesco Bonchi, Amit Goyal, and Laks V. S. Lakshmanan. Viral Marketing Meets Social Advertising: Ad Allocation with Minimum Regret. *Proc. VLDB Endow.*, 8(7):822–833, 2015.
[6] Roja Bandari, Sitaram Asur, and Bernardo A Huberman. The pulse of news in social media: Forecasting popularity. In *Proc. Intl. AAAI Conf. on Weblogs and Social Media (ICWSM)*, 2012.
[7] Cody Buntain and Jennifer Golbeck. Automatically Identifying Fake News in Popular Twitter Threads. In *Proc. IEEE Intl. Conf. on Smart Cloud (SmartCloud)*, pages 208–215. IEEE Computer Society, 2017.
[8] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. Popularity Prediction on Social Platforms with Coupled Graph Neural Networks. In *Proc. ACM Intl. Conf. on Web Search and Data Mining WSDM*, pages 70–78. ACM, 2020.
[9] David Dupuis, Cédric du Mouza, Nicolas Travers, and Gaël Chareyron. RTIM: A Real-Time Influence Maximization Strategy. In Reynold Cheng, Nikos Mamoulis, Yizhou Sun, and Xin Huang, editors, *Proc. Intl. Conf. on Web Information Systems Engineering (WISE)*.
[10] Manish Gaurav, Amit Srivastava, Anoop Kumar, and Scott Miller. Leveraging candidate popularity on twitter to predict election outcome. In *Proc. Intl. Work. on Social Network Mining and Analysis*, pages 1–8, 2013.
[11] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proc. ACM Intl. Conf. on Knowledge Discovery and Data mining (KDD)*, pages 355–359, 2000.
[12] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proc. Intl. Conf. on Data Engineering (ICDE)*,

pages 215–224, 2001.
[13] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.
[14] Liangjie Hong, Ovidiu Dan, and Brian D Davison. Predicting Popular Messages in Twitter. In *Proc. Intl. Conf. on World Wide Web (WWW)*, pages 57–58, 2011.
[15] Sounman Hong and Daniel Nadler. Which candidates do the public discuss online in an election campaign?: The use of social media by 2012 presidential candidates and its impact on candidate salience. *Government information quarterly*, 29(4):455–461, 2012.
[16] Imran N. Junejo and Zaher Al Aghbari. Using SAX representation for human action recognition. *Journal of Visual Communication and Image Representation*, 23(6):853–861, August 2012.
[17] Amir Karami and Aida Elkouri. Political Popularity Analysis in Social Media. *CoRR*, abs/1812.03258, 2018.
[18] David Kempe, Jon Kleinberg, and Eva Tardos. Maximizing the Spread of Influence through a Social Network. *Theory of Computing*, 11:43, 2015.
[19] Paul Lagree, Olivier Cappe, Bogdan Cautis, and Silviu Maniu. Effective Large-Scale Online Influence Maximization. pages 937–942. IEEE, November 2017.
[20] Hui Li, Sourav S. Bhowmick, and Aixin Sun. *CASINO: Towards Conformity-aware Social Influence Analysis in Online Social Networks*.
[21] Hui Li, Sourav S. Bhowmick, and Aixin Sun. CINEMA: conformity-aware greedy algorithm for influence maximization in online social networks. page 323. ACM Press, 2013.
[22] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, August 2007.
[23] Witold Litwin. Linear Hashing: A New Tool for File and Table Addressing. In *Intl Conf. on Very Large Data Bases (VLDB)*, pages 212–223, 1980.
[24] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Rt to Win! Predicting Message Propagation in Twitter. In *Proc. Intl. AAAI Conf. on Weblogs and Social Media (ICWSM)*, 2011.
[25] Ramakrishnan Srikant. *Fast algorithms for mining association rules and sequential patterns*. PhD thesis, Citeseer, 1996.
[26] Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.
[27] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. TwitterRank: finding topic-sensitive influential twitterers. page 261. ACM Press, 2010.
[28] Yuto Yamaguchi, Tsubasa Takahashi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. TURank: Twitter User Ranking Based on User-Tweet Graph Analysis. In Lei Chen, Peter Triantafillou, and Torsten Suel, editors, *Intl. Conf. on Web Information Systems Engineering (WISE)*, pages 240–253, 2010.
[29] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.
[30] Tauhid R Zaman, Ralf Herbrich, Jurgen Van Gael, and David Stern. Predicting information spreading in twitter. In *Proc. Intl. Work. on Computational Cocial Science and the Wisdom of Crowds*, volume 104, pages 17599–601, 2010.
[31] Zilong Zhao, Jichang Zhao, Yukie Sano, Orr Levy, Hideki Takayasu, Misako Takayasu, Daqing Li, Junjie Wu, and Shlomo Havlin. Fake News Propagates Differently from Real News even at Early Stages of spreading. *EPJ Data Sci.*, 9(1):7, 2020.